RESEARCH PAPER

Realising Data-Centric Scientific Workflows with Provenance-Capturing on Data Lakes

Hendrik Nolte[†] **& Philipp Wieder**

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen Göttingen, Gottingen 37077, Germany

Keywords: Data lake; Provenance; Workflows; FAIR Digital Objects; CWFR

Citation: Nolte, H., Wieder, P.: Realising data-centric scientific workflows with provenance-capturing on data lakes. Data Intelligence 4(2), 426-438 (2022). doi: 10.1162/dint_a_00141

Received: September 18, 2021; Revised: November 26, 2021; Accepted: Feburary 5, 2022

ABSTRACT

Since their introduction by James Dixon in 2010, data lakes get more and more attention, driven by the promise of high reusability of the stored data due to the schema-on-read semantics. Building on this idea, several additional requirements were discussed in literature to improve the general usability of the concept, like a central metadata catalog including all provenance information, an overarching data governance, or the integration with (high-performance) processing capabilities. Although the necessity for a logical and a physical organisation of data lakes in order to meet those requirements is widely recognized, no concrete guidelines are yet provided. The most common architecture implementing this conceptual organisation is the zone architecture, where data is assigned to a certain zone depending on the degree of processing. This paper discusses how FAIR Digital Objects can be used in a novel approach to organize a data lake based on data types instead of zones, how they can be used to abstract the physical implementation, and how they empower generic and portable processing capabilities based on a provenance-based approach.

1. INTRODUCTION

Data pipelines are widely used (i) in order to collect data from heterogeneous sources, (ii) to perform a series of transformations on them, and (iii) to ingest the transformed data into a destination system for data analytics. Such a data pipeline can also be referred to as an ETL (extract, transform, load) process, which is often used to ingest clean and transformed data into a data warehouse [1]. Data pipelines are still

[†] Corresponding author: Hendrik Nolte (Email: hendrik.nolte@gwdg.de; ORCID: 0000-0003-2138-8510).

considered to be state-of-the-art, although several draw-backs are well known [2]. One of these problems is the information loss during the ETL process, which can, for instance, happen when the most relevant attributes are being aggregated in order to make the format suitable for a data warehouse. Consequently, only a predetermined subset of the attributes is available for a subsequent analysis. Thus, the schema-on-write semantics of this approach limits the reuse of data stored in traditional data management systems for analytics outside of the original scope. In order to prevent this information loss, the concept of data lakes was introduced by Dixon in 2010 [3]. A data lake, opposed to a data warehouse, is generally designed as the central repository for all data sets from all data sources in their raw format [4]. Since no transformation is needed to ingest data into a data lake and no assumptions are made about subsequent analysis, a schema-on-read approach is used which ensures high flexibility and re-usability.

Although no commonly accepted concept for a data lake exists in literature [5], an agreed upon outline, as it was described by Dixon, requires a scalable storage system for heterogeneous data where scientists can explore and analyze these data sets. These requirements go hand in hand with the need for low-cost technologies and at first led to a strong association of data lake implementations with Apache Hadoop [6]. This was then superseded by proprietary cloud solutions based on Azure or AWS [4], which introduced the advantage of separating storage and compute resources into the concept of data lakes.

Although a data lake implements a schema-on-read semantic, some modeling is mandatory to ensure proper data integration, comprehensibility, and quality [7]. Such data models, created by extracting descriptive metadata from the ingested raw data, are then stored in a central data catalog. This data catalog does not enforce a fixed schema and therefore allows for frequent changes [8]. Although many studies primarily focus on this data catalog and the gradual improvement of the provisioned metadata [9, 10], such as semantic information about a data set [11], provenance data is not yet extensively covered by current approaches. However, since the raw data in a data lake is very likely subjected to many consecutive transformations, resulting in several artefacts, which will be ingested back into the data lake, maintaining concise provenance information is very challenging but crucial to retain the manageability of a data lake [12]. Processed data, which is just being stored back into the data lake, will be hard to find afterwards and probably impossible to comprehend and to reproduce, potentially rendering it useless.

In this paper we discuss (i) how the strong association between a single data entity and its associated metadata can be expressed within a FAIR Digital Object [13] in particular within the context of the data lake, (ii) how the usage of typed digital objects can conceptually organize a data lake architecture, (iii) how packaging completely generic analysis tasks within Fair Digital Objects can help to monitor fain grained provenance information, and (iv) how exploitation of these concepts in the context of Canonical Workflow Frameworks for Research (CWFR)[①] can help to increase cross discipline collaboration.

[①] https://osf.io/9e3vc/

2. RELATED WORK

Various solutions tailored for specific purposes have been proposed for provenance capturing in data lakes. *Goods* [14] analyses log files in a post-hoc manner to determine which jobs created data sets based on which input, an approach which requires that the application writes suitable log files. Similarly, *Komadu* was integrated into a data lake [12] to support the messaging of provenance information via RabbitMQ, also relying on the explicit support of the application. *DataHub* [15] was equipped with *ProvDB* [16] for provenance auditing. *ProvDB* is based on the analysis of shell scripts, user annotations, post-hoc log analysis from frameworks like *Caffe* and table-based input files for SQL-based provenance capture. This approach, however, is error-prone and the monitoring of system calls introduces additional run-time overhead.

An common data lake architecture is the so-called *zone architecture* [5, 17]. The general idea is to divide the data lake into different zones and store data depending on the amount of processing it was subjected to. In this way, raw data is stored in a *raw zone* whereas (pre-)processed data are stored in their own dedicated zones. Although there has recently been great progress in the definition of a standardized zone reference model[18], this architecture lacks a homogeneous and standardized interface to integrate processes and users, a central data catalog to maximize re-usability, and a reliable provenance auditing by design.

It is by now common sense that scientific publications are be complemented with an organized aggregation of all related digitized entities, in order to promote comprehensibility of the presented work and further reusability of the data and methods. *Research Objects* [19] have been proposed as an structured aggregation, which semantically links its entities. It also defines a standard regarding repetition, traceability, and the essential metadata. Different tools, like *Sciunits* [20], have been developed to automate the process of building a research object from a workflow enactment. Here, application virtualization, based on the monitoring of system calls, is used to build one ready-to-use container including all software, its dependencies and the used data.

All of these above mentioned developments tackle independently of each other different problems in the area of collaborative data analytics. However, the full potential can only be realized if these individual steps are set into relation to each other and are provided to the users as an integrated solution.

3. IMPLEMENTATION

In this section, we present a data lake implementation that is particularly tailored to be used in data and compute intensive research areas. We put a special focus on the traceability and reproducibility of the different processing steps, like data ingestion or analysis operations. Since the presented data lake implementation is a generic framework aiming at supporting various use cases, we separate the implementation of the aforementioned processing steps from the data lake implementation and thus make it configurable and extendable for individual use cases. In addition, the data lake includes a central role-based user access management, which supports scalability and large teams.

3.1 Architecture Overview

The core our data lake implementation is the central web application, which i.a. provides the REST-API of the data lake. It also implements the orchestration functions, so that a consistent state of the data lake and the essential minimum of (meta)data quality is being enforced. Here, the actual mapping between the logical and the physical organization of the data lakes happens.

A schematic view is provided in Figure 1. Here, the web application as the single point of contact for data sources to ingest data, but also for users to interact with the data lake is shown. That means, that all services which are needed to realize the complete data lake are not directly accessed by the users, but those are always situated behind the web application. Therefore these back-end services can be easily swapped or added without requiring any adoptions by the users for existing workflows and pipelines. There is, however, one exception: The direct interaction of users with version control tools, like *GitLab*, is tightly integrated into our concept of a data lake in order to support widely used development workflows.

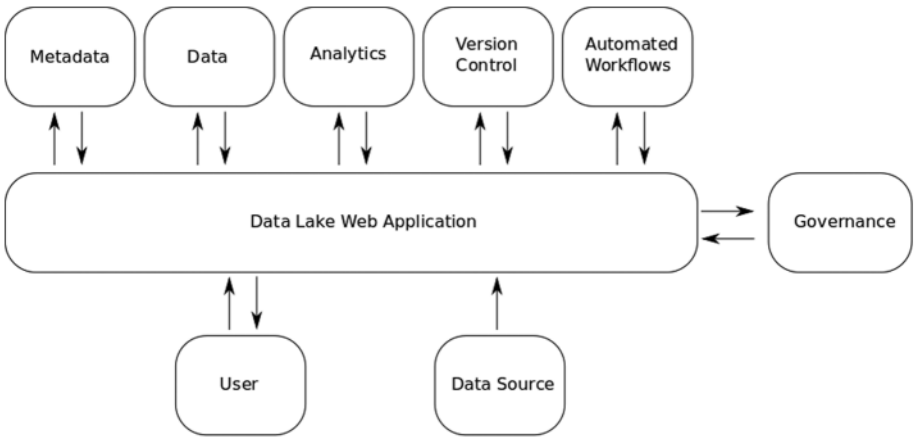

Figure 1. Schematic view of the implemented data lake architecture. The organization of the different underlying services is abstracted by the unifying data lake layer where an overarching governance is enforced.

Another advantages of this data lake implementation, in addition to scalability, consistency, and maintainability, are on the one hand the fine grained control and on the other hand the high-level abstractions of the underlying processes, which empowers non-experts to use the data lake.

3.2 Ingestion

It is widely accepted that a data lake, as the central data repository of an institution, should accept any data, i.e. any type and format, from any source [4]. Therefore, the ingestion layer of a data lake has to be highly flexible. This has in particular implications on two parts. First, the data transmission should use common protocols like http. Here, it can be even favorable if a data lake is not only a passive data sink, but can actively pull data from other systems in order to stay up-to-date. This can also reduce maintenance overhead, since a lot of different and possibly heterogeneous data pipelines can be managed from a single system and interface. Second, the storage and metadata systems need to be able to cope, or at least easily adopt to, any kind of input data in order to be able to automatically extract descriptive metadata to update the central data catalog. In order to be able to upload a new file containing raw data, the user first has to configure a new metadata type, containing the key-value pairs with their respective mappings. Then, another API of the data lake is used to upload containers, which contain a metadata extraction tool, which can be e.g. a simple *Python* script. The kind of metadata that should be extracted at this point from the ingested raw data can be categorized into the *pre-visualization and summary* and *semantic* metadata [10], which primarily contain informative and descriptive information.

3.3 Data Analytics

After the raw data has been ingested into the data lake, the next step is to transform and analyse these data sets. To achieve this, a job manifest has to be written, which unambiguously describes the job that has to be performed. This job manifest is then being interpreted by a dedicated adapter and the defined computation is performed. Currently, two different adapters have been implemented to execute a job either by using local resources or by remotely connecting to a HPC system using *HPCSerA* [21]. New adapters can easily be defined and thus new compute environments can be added, like cloud-based function-as-a-service (FaaS). Due to the compute intensity of most analytics tasks, outsourcing to an HPC-System is preferred over local computation. In order to execute an job on an HPC system, first some pre-processing is needed to setup the environment, exactly as specified in the job manifest. Then, the run script is submitted to the HPC batch system, where the actual computation is being performed. The entire environment with all its dependencies is completely virtualised within a container. In order to guarantee reproducibility, container images are uploaded into the data lake beforehand, and will be retained there as long as there are processed data linking to that particular image. These images are represented as FAIR Digital Objects, like the job manifests and all kinds of data and all other entities. In addition, only input data that has been specified in the job manifest are available to the analytics job. Since all parts of the computations are therefore completely encapsulated, there is on one hand a high portability and on the other hand no need for run time provenance auditing, since all possible degrees of freedom are either specified in the job manifest or are determined during the pre-processing (like the executed git commits).

After the computation some post-processing is performed, where data artifacts, which result from the analysis, can be ingested back into the data lake. There are two different options from which the user can choose: (i) data are ingested simply as an processed data type, thus only having descriptive metadata in form of their retrospective provenance information, or (ii) the manual ingestion process as described in Section 3.2 is executed again. In the latter case descriptive metadata, in addition to automatically captured and attached retrospective provenance information, can be extracted to allow for queries based on the content and not only on the data lineage.

3.4 Constantly Evolving Metadata

In order to achieve a consistent metadata quality despite a constantly evolving metadata schema, a rather simple modeling approach is being taken. First, four top level data types are defined, *raw data*, *processed data*, *container* and *manifest*, from which other data types can be derived. Users can register derived data types by defining a new schema which consists of the associated attributes and an assigned name for the new data type. Through this, only known data types with known and typed attributes are being indexed and ingested into the data lake. Based on this environment of well controlled attributes one can start to maintain an data lake wide ontology, defining categories, the associated properties and relations between those data types and their attributes.

By versioning the data types, continuous change is enabled and, at the same time, the necessary provenance information is maintained to keep full control of the state of the data lake. Performing data analytics on a raw data type and re-ingesting a processed data type automatically causes the data lake to build up a data provenance-centered graph model. Here, the derived data entities, i.e. the aggregated data and metadata, are connected to their individual input data by the job manifest.

4. EXAMPLE WORKFLOW

To apply our generic data lake implementation to a particular, project-specific use case, users first need to perform several configuration steps (see also Section 3.2). First of all, some data modelling is required resulting in a *JSON* document specifying a metadata template of a FAIR Digital Object. This template contains the typed attributes, which a raw or processed data type can have, the type of the FAIR Digital Object itself, and some information to derive a Digital Object Identifier. Afterwards, a metadata extraction container needs to be uploaded to the data lake. A subsequent configuration is necessary to configure the execution command and related options. At this point everything is prepared to upload data as the previously defined FAIR Digital Object. As a result of the upload, the metadata extraction container automatically extracts the metadata, which itself is added to the data lake in *JSON* format and checked for consistency. Finally, the metadata is indexed in a NoSQL data base, like ElasticSearch, and the actual date, i.e. the actual bit sequence, can be saved in a scalable storage like an S3-based object store.

Now, analyses can be performed on this data set, in this case on an HPC system. Analog to the metadata extraction container, an analysis container image is uploaded to the data lake. Assuming that the necessary adapter, like *HPCSerA*, is configured for the particular user, a job manifest can be send to the data lake to manually trigger the execution of an analysis job (see also Section 3.3). The respective container image is then copied to the HPC system along with the input data. If the container does not ship with the necessary software, a *git* repository can be dynamically cloned and build. All necessary information like the used commit or build commands are captured. After processing, specified artefacts are automatically ingested into the data lake. Here, based on a user provided configuration, the data lake will either perform the full ingestion process to create a Digital Object of a specific type from these artefacts, or they will be be ingested as a generic *Processed Data*-Type. In the latter case, no descriptive metadata except the provenance information will be available.

Currently, all provenance information is written as *JSON* and indexed in ElasticSearch, but a refined model compliant with the W3C PROV specification[②] using a graph database is under development.

5. TOWARDS A GENERIC DATA ANALYTICS PIPELINE

The current implementation of the data lake exclusively covers the data processing part and therefore fulfils the requirements of the primary use case. More complex research workflows, which may e.g. include experiments, can be mapped onto the data lake by adding elaborate descriptions of experimental protocols to the metadata of the measured data or by uploading the experimental protocol itself as raw data into the data lake and linking it to the corresponding data sets. In order to support collaborations there is the need to share developed analysis tools. A convenient way to do so is by providing containers. Here, we go one step further and allow for a precise configuration which enables machine actionability on the available data. Since scientific workflows usually consist of a sequence of steps, there is a need to link single tasks to a workflow.

5.1 Publishing Analysis Tasks

The general analysis process as described in Section 3.3 is highly flexible and supports the rapid development of the involved algorithms and software. Once such a software has reached maturity, the developers usually publish it. This process can be completely achieved within the data lake. Here, similar to a job manifest, a slightly changed analysis service needs to be defined. This service can then be used by other data lake users by uploading suitable data into the data lake and starting this analysis service on it. For this process the access policies of the uploaded data can be completely private. The analysis service is accessed by its own unique resource identifier which can be resolved by the data lake. Although using such published analysis services will be fairly easy for other users, since most of the configuration has been done by the actual developers, such an analysis service should be able to provide, upon request, a minimum set of documentation of the available options. Here, particularly the specification of suitable input data is of out most importance.

5.2 Linking Tasks to Workflows

In order to obtain an automated workflow these single tasks need to be chained together. Since the essential element of a data lake is the data itself, it seems reasonable to focus on data centric workflows which are often represented by a directed acyclic graphs. Having all the necessary task definitions in place, i.e. the generalized job manifests as described in Section 5.1, the independent tasks simply need to be linked by their respective input and output data. This can be done in an workflow manifest, where tasks should be either generalized job manifests or published analysis tasks. Here, the data lake functions again as an additional abstraction layer between the workflow definition and the actual workflow implementation.

[②] https://www.w3.org/TR/prov-overview/

This requires adapters to map the workflow manifest onto suitable workflow engines, like CWL [22] or Apache Airflow. These mappings also support portability outside of the context of the data lake to re-run and verify a workflow locally. However, the intermediary workflow manifest concepts makes the data lake highly adaptable as it enables tailor-made solutions for individual projects while still enforcing well-defined and homogeneous provenance information and artefact handling.

6. DATA ANALYTICS PIPELINE USING FAIR DIGITAL OBJECTS AND CWFR

Expressing all entities in a data lake as FAIR Digital Objects (DO) is a novel approach, which provides a practical guideline to become compliant with the general requirement that a data lake needs to have a logical and a physical organization [4]. This is realised through direct user interaction with the different FAIR Digital Objects by calling functions that are specifically defined for the corresponding type. Furthermore, all FAIR DOs have their own governance, which can be integrated into a data lake-wide homogeneous governance layer to realise high scalability across different back-end services.

6.1 Example Interaction with FAIR Digital Objects

As previously stated, the fundamental building block of our data lake architecture are FAIR Digital Objects. Here, users do not work directly on the services, but call functions which are part of the specific Digital Object instance. This process is discussed by an example of working with an metadata extraction container.

First of all, a user registers a Digital Object Identifier that resolves to the metadata extraction container. On this level, it is an encapsulation that can contain multiple images and configurations. This step also creates an corresponding object representation in the NoSQL database, where information about every Digital Object is kept. Then, a specific container image is uploaded into this Digital Object, using a predefined upload function of that instance. Although this image is itself a Digital Object again, we will not discuss it as it is mostly immutable. The previous function call has stored the container image in our backend-storage and has added the particular image to the record of the metadata extraction container Digital Object. This record is part of the metadata of this Digital Object and is currently implemented as a SQL table. Part of this record are also image-specific configurations, like an execution command, or an optional bind mount. Since the exact configuration is part of the provenance information of the ingested data, a function to update the configuration can be exposed without loosing the ability to reproduce the results. An update image function is able to add a new image to the Digital Object representing a metadata extraction container with a new configuration. If this function is used, the Digital Object will contain more then one image, with the different configurations logged in the record. A delete function, defined on both the wrapping metadata extraction Digital Object and the image Digital Object can only be executed, if this particular Digital Object Identifier is not present in any provenance information, to ensure reproducibility.

In this example, users only interact with the predefined functions of the Digital Object trough the REST API exposed by the web application. There is no need that users interact directly with the underlying database or storage systems. This ensures a consistent state across the entire data lake.

6.2 Providing Canonical Steps

As described in Section 5.1, data scientists and solution developers can provide their analysis and metadata extraction tools to other users of the data lake. To achieve this, a more generic service manifest is required, which primarily deviates from a job manifest that input data is implicitly selected by specifying the data types and metadata attributes of the input data to restrict the possible input data to a suitable subset. In a job manifest input data is selected explicitly by lists or queries. Depending of the configured specification of those typed metadata attributes and their exact value, fine grained conditions can implemented to control the execution of such a service. Since this process is relying on typed attributes, it can be automated and interoperability between the data and the analysis tasks is achieved. Furthermore, those services can be encapsulated in an FAIR DO and functions can be associated with them. Since data scientists can develop and share those pre-configured analysis and metadata extraction tasks, which are the canonical steps of a data lake workflow, we foster re-usability, sharing, and collaborations.

6.3 Encapsulation and Aggregations of Data and Workflows

Upon the ingestion of raw data into the data lake, data is associated with a certain data type and as such encapsulated with the associated metadata. The actual data, and as such their associated bit sequence, which is physically stored on any suitable storage system, and the extracted metadata which can be split into different logical sub-units and be indexed individually in different database systems, are made resolvable by a single unique resource identifier. The information about the physical organization can be completely abstracted from the user by only allowing interactions on the data entities by specific services, which promotes the objectification of data. Important in the context of a data lake is that the association of a data type to the raw data must not lead to a fixed schema but integrates smoothly with the required data catalog. Further aspects, like the access control lists, which might be defined on an DO, also need to be integrated in the overarching governance layer of a data lake. Based on these implicitly build data objects a data lake wide ontology enables an automated classification of this new data into the context of the data lake. Further analysis tasks can add more metadata and context to the data entity.

The beginning of a data science project is often marked by an ad-hoc data exploration phase. Particularly here, users often neglect development and analysis workflows, which enable reproducibility, in favor for development time. Treating data and its associated metadata strictly as a unified object eases the way to integrate ad-hoc data science capabilities. Here, users can work interactively on their data in the data lake by using a library which imports the data objects from the data lake into their programming environment. The challenge is to have a provenance monitoring system transparently running in order to be able to store resulting artifacts with their respective provenance data in the data lake. One example to achieve this would be to transparently tap into the data collected by tools like *ProvBook* [23] and send the collected data along with the artifact itself when it is being saved. This artifact, along with its metadata, is then an object itself in the data lake. However, it is desirable to access the entire workflow as a whole. For this arbitrary aggregations of the different entities of the data lake are necessary. Analog to this example, the requirement of aggregations also persist in automated workflows where raw data should be bundled with a workflow manifest, the resulting artifacts, the used environment and the job manifests in order to promote reproducibility and reusability [21]. This can be done completely in metadata space since upon performing an analysis a provenance-centered graph model is automatically constructed. Here, a simple backtraversion towards the raw data of all nodes will guarantee to yield all involved entities. Aggregating them into a single wrapper entity represents the exact state of a workflow. In order to work more easily with those entities, the general taxonomy of the data types of the data lake can be used to offer the user an overview and access to the constituents in the desired granularity. Since every action is already abstracted, e.g. performing an analysis is defined in a job manifest, the concept of adapters can be reused to enable portability of those aggregations and re-execute tasks on different systems using different software stacks. In order to promote a high interoperability, the data lake should export those aggregations in well established formats, like research objects [24]. Here we envision standardized operations which can be performed on such aggregation, like inspecting the different entities or re-executing the workflow or singular steps.

6.4 Challenges

The analysis of the current data lake approach identified two challenges which resulted from our explicit implementation, but may also require a more formal consideration in general.

Automating workflows, using FAIR Digital Objects to achieve interoperability between the data and the analysis tasks, have the inherent risk of the difficulty to control data multiplication and the corresponding surge in compute demand. Assuming an potentially ill-configured analysis task, which takes in n different data types and outputs m different artifacts. If a new data source would be made available to the data lake resulting in a huge increase in available entities of the before mentioned n data types. In case this automatically triggers innumerable compute and storage intensive analysis tasks on the data lake, although the author of this analysis task never intended the use on such a big data set, the resources of the data lake, or the personal quotas, can be quickly reached, entailing an undesirable interruption of the service.

Currently wrappers, called manifests, are used to abstract the definition and description of a task or workflow from the underlying software stack. In order to use the underlying local software stack, adapters are needed which map the abstract instructions into the language of specific tools. This entails the necessity to develop and maintain a number of adapters. This problem can only partly be mitigated by using a suitable inheritance hierarchy. In addition, the different software stacks differ in functionality. Thus, it is not possible to support all state-of-the-art features of one technology and still offer the same amount of portability, as compared to a manifest using a stripped-down-version of the available functionalities.

7. CONCLUSION

In conclusion, we present a novel data lake architecture, which is based on typed digital objects. Our approach shows significant advantages as it allows to easily adapt the architecture by defining new data types and offers a uniform and research oriented user and process interface while keeping a certain technology independence. It is important to stress that no prior assumptions about the specific use case are made which will maximize re-usability of the ingested data, also supported by maintaining a central data catalog which contains all data independent of the degree of the processing. All FAIR Digital Objects are linked in a single provenance graph, from the raw data to the final product. The additional abstraction layers, i.e. the manifests and the corresponding adapters, make our data lake highly portable and provide homogeneous retrospective provenance auditing independent of the chosen workflow engine or computational environment. Furthermore, representing all entities of the data lake as objects supports the integration into application- or user-specific processes, starting with an ad-hoc data exploration phase towards fully automated workflows. Here, data analytics tasks can be packaged as services itself, facilitating exchange and collaboration across teams and scientific domains.

ACKNOWLEDGEMENTS

We acknowledge funding by the “Niedersächsisches Vorab” funding line of the Volkswagen Foundation.

AUTHOR CONTRIBUTION STATEMENT

Hendrik Nolte (hendrik.nolte@gwdg.de): Conceptualization, Software, Writing—Original draft.

Philipp Wieder (philipp.wieder@gwdg.de): Writing—Review editing, Supervision.

REFERENCES

[1] Ali El-Sappagh, S.H., Ahmed Hendawi, A.M., El Bastawissy, A.H.: A proposed model for data warehouse ETL processes. Journal of King Saud University—Computer and Information Sciences 23(2), 91–104 (2011)

[2] Munappy, A.R., Bosch, J., Olsson, H.H.: Data pipeline management in practice: Challenges and opportunities. In: Morisio, M., Torchiano, M., Jedlitschka, A. (eds). Product-Focused Software Process Improvement, pp. 168–184. Springer, Cham (2020)

[3] Dixon, J.: Pentaho, Hadoop, and data lakes. Available at: ttps://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. Accessed 5 February 2022

[4] Madera, C., Laurent, A.: The next information architecture evolution: The data lake wave. In: Proceedings of the 8th International Conference on Management of Digital EcoSystems, pp. 174–180 (2016)

[5] Giebler, C., et al.: Leveraging the data lake—current state and challenges. DaWaK 2019: Big Data Analytics and Knowledge Discovery, pp. 179–188 (2019)

[6] Khine, Pwint Phyu and Wang, Zhao Shun. Data lake: a new ideology in big data era. *ITM Web Conf.*, 17, 03025 (2018)

[7] Hai R., Quix C., Zhou C. Query Rewriting for Heterogeneous Data Lakes. *IAdvances in Databases and Information Systems*, 11019 (2018)

[8] Mathis, C. Data Lakes. *Datenbank Spektrum*, 17 (2017)

[9] Hai, Rihan and Geisler, Sandra and Quix, Christoph. Constance: An Intelligent Data Lake System. pages 2097–2100, 2016.

[10] Sawadogo, Pegdwendé Nicolas and Scholly, Etienne and Favre, Cécile and Ferey, Eric and Loudcher, Sabine and Darmont, Jérôme. Metadata Systems for Data Lakes: Models and Features. 2019.

[11] Hai, Rihan and Quix, Christoph and Zhou, Chen. Query Rewriting for Heterogeneous Data Lakes. In Benczúr, András and Thalheim, Bernhard and Horváth, Tomáš, editors, *Advances in Databases and Information Systems*, pages 35–49, Cham, 2018. Springer International Publishing.

[12] Suriarachchi, Isuru and Plale, Beth. Crossing analytics systems: a case for integrated provenance in data lakes. *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 349–354, 2016. IEEE.

[13] De Smedt, Koenraad and Koureas, Dimitris and Wittenburg, Peter. FAIR Digital Objects for Science: From Data Pieces to Actionable Knowledge Units. *Publications*, 8(2), 2020.

[14] Halevy, Alon Y and Korn, Flip and Noy, Natalya Fridman and Olston, Christopher and Polyzotis, Neoklis and Roy, Sudip and Whang, Steven Euijong. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.*, 39(3): 5–14, 2016.

[15] Bhardwaj, Anant and Bhattacherjee, Souvik and Chavan, Amit and Deshpande, Amol and Elmore, Aaron J and Madden, Samuel and Parameswaran, Aditya G. Datahub: Collaborative data science & dataset version management at scale. *arXiv preprint arXiv:1409.0798*, 2014.

[16] Miao, Hui and Chavan, Amit and Deshpande, Amol. Provdb: Lifecycle management of collaborative analysis workflows. *Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics*, pages 1–6, 2017.

[17] Sawadogo, Pegdwendé and Darmont, Jérôme. On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, 56(1): 97–120, 2021.

[18] Giebler, C., et al.: A zone reference model for enterprise-grade data lake management. In: 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), pp. 57–66 (2020)

[19] Bechhofer, Sean and De Roure, David and Gamble, Matthew and Goble, Carole and Buchan, Iain. Research Objects: Towards Exchange and Reuse of Digital Knowledge. *Nature Precedings*, 2010.

[20] Dai Hai Ton That and Gabriel Fils and Zhihao Yuan and Tanu Malik. Sciunits: Reusable Research Objects. *CoRR*, abs/1707.05731, 2017.

[21] Bingert, Sven and Köhler, Christian and Nolte, Hendrik and Alamgir, Waqar. An API to Include HPC Resources in Workflow Systems. *INFOCOMP 2021: The Eleventh International Conference on Advanced Communications and Computation*, pages 15–20, 2021.

[22] Amstutz, P., et al.: Common workflow language, v1. 0 (2016). Available at: https://figshare.com/articles/dataset/Common_Workflow_Language_draft_3/3115156/2. Accessed 5 February 2022

[23] Samuel, Sheeba and König-Ries, Birgitta. ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. *International Semantic Web Conference (P&D/Industry/BlueSky)*, 2018.

[24] Chard, Kyle and D'Arcy, Mike and Heavner, Ben and Foster, Ian and Kesselman, Carl and Madduri, Ravi and Rodriguez, Alexis and Soiland-Reyes, Stian and Goble, Carole and Clark, Kristi and Deutsch, Eric W. and Dinov, Ivo and Price, Nathan and Toga, Arthur. I'll take that to go: Big data bags and minimal identifiers for exchange of large, complex datasets. *2016 IEEE International Conference on Big Data (Big Data)*, pages 319–328, 2016. IEEE.

AUTHOR BIOGRAPHY

Hendrik Nolte obtained his M.SC. Physics from the University of Göttingen and has been working since November 2019 as a research assistant at the Gesellschaft für wissenschaftliche Datenverarbeitung mbH. He is working together with different institutes to build data lakes tailored to their specific needs. His particular interests lie in secure processing of sensitive data and provenance auditing. He is pursuing a Ph.D. in the field of data lakes and is collaborating for this with the University's Medical School in a common research project to improve MR image processing.
ORCID: 0000-0003-2138-8510

Philipp Wieder received his Ph.D. degree from TU Dortmund, Germany. He is the Deputy Head of the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Germany, and is teaching Data Science Infrastructures at the Georg-August-Universität Göttingen, Germany. He is contributing to areas like distributed infrastructures, research data management, and data analytics for several years. His research interest lies in distributed systems, service level agreements, and resource scheduling. He is actively involved in the German National Research Data Initiative (NFDI) and several projects related to the European Open Science Cloud.
ORCID: 0000-0002-6992-1866